

CLEAN CODE

WRITING CODE

Writing Code

Agenda

- ✓ Naming conventions
- ✓ Functions
- ✓ Comments
- ✓ Formatting

NAMING CONVENTIONS

Naming conventions

Basic rule



Classes and Objects

Noun

Noun phrase



Variables, Properties, Parameters

Noun



Methods and Functions

Verb

Verb phrase

Naming Conventions

Use Intention-Revealing Names

- ✓ Names should reveal intent
- ✓ Don't name a variable, class, or method which needs some explanation in comments

```
1  
2  int d; // elapsed time in days
```



```
1  
2  int elapsedTimeInDays;
```

- ✓ Should tell: **WHY** it exists, **WHAT** it does, and **HOW** it is used

```
1  
2  for (int j=0; j<34; j++) {  
3      |   s += (t[j]*4)/5;  
4  }
```



```
1  
2  int realDaysPerIdealDay = 4;  
3  const int WORK_DAYS_PER_WEEK = 5;  
4  int sum = 0;  
5  for (int j=0; j < NUMBER_OF_TASKS; j++) {  
6      |   int realTaskDays = taskEstimate[j] * realDaysPerIdealDay;  
7      |   int realTaskWeeks = (realdays / WORK_DAYS_PER_WEEK);  
8      |   sum += realTaskWeeks;  
9  }
```

Naming Conventions

Avoid Disinformation

- ✓ Avoid leaving false clues that obscure the meaning of code

```
public List  
public Collection  
public SparkPlug
```

```
sparkPlugs  
sparkPlugList  
sparkPlug
```



```
public List  
public Collection  
public SparkPlug
```

```
sparkPluglist  
sparkPlugCollection  
currentSparkPlug
```

Naming Conventions

Make Meaningful Distinctions

- ✓ If something mean different, then the names must be different
- ✓ Avoid using noise word such as “ProductInfo” or “ProductData”

```
public String productInfo  
public String productDetails
```



```
public String productName  
public String productDescription
```


Naming Conventions

Use Pronounceable And Searchable Names

- ✓ Make your names pronounceable
- ✓ Make your names Searchable
- ✓ Avoid encoding

```
class DtaRcrd102 {  
    private Date genymdhms;  
    private Date modymdhms;  
    private final String pszqint = "102";  
};
```



```
class Customer {  
    private Date generationTimestamp;  
    private Date modificationTimestamp;  
    private final String recordId = "102";  
};
```

Naming Conventions

Take away

Choosing good names takes time but **saves** more than it takes.

✓ Classes and Objects	Noun	Noun phrase
✓ Variables, Properties, Parameters	Noun	
✓ Methods and Functions	Verb	Verb phrase

- ✓ Intention-**Revealing** Names
- ✓ **Avoid** Disinformation
- ✓ Meaningful **Distinctions**
- ✓ Pronounceable
- ✓ Searchable

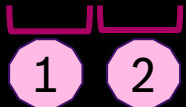
FUNCTIONS

Functions

Small

- ✓ **Small** and should be **smaller**
- ✓ Indent level of a function should not be greater than one or two

```
public static String renderPageWithSetupsAndTeardowns(  
    PageData pageData, boolean isSuite) throws Exception {  
    if (isTestPage(pageData))  
        includeSetupAndTeardownPages(pageData, isSuite);  
    return pageData.getHtml();  
}
```



Functions

Do One Thing



Should do one thing, do it well, do it only

```
public static boolean isCurrentUserInRole(String authority) {  
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();  
    Stream<String> authorities = authentication.getAuthorities().stream().map(GrantedAuthority::getAuthority);  
    return authentication != null &&  
        authorities.anyMatch(authority::equals);  
}
```



```
public static boolean isCurrentUserInRole(String authority) {  
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();  
    return authentication != null &&  
        getAuthorities(authentication).anyMatch(authority::equals);  
}  
  
private static Stream<String> getAuthorities(Authentication authentication) {  
    return authentication.getAuthorities().stream()  
        .map(GrantedAuthority::getAuthority);  
}
```

Functions

One Level of Abstraction per Function

- ✓ Statements within our function are all at the same level of abstraction
- ✓ Read code from top to bottom

Abstraction Level 1

Abstraction Level 2

```
public static void main(String[] args) {  
    Application app = new Application(EtravelApp.class);  
    DefaultProfileUtil.addDefaultProfile(app);  
    Environment env = app.run(args).getEnvironment();  
    logApplicationStartup(env);  
}  
  
public static void addDefaultProfile(Application app) {  
    Map<String, Object> defProperties = new HashMap<>();  
    defProperties.put(APP_PROFILE_DEFAULT, AppConstants.APP_PROFILE_DEVELOPMENT);  
    app.setDefaultProperties(defProperties);  
}
```

Abstraction Level 3

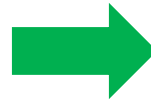
Functions

“One Switch” rule



Should use **polymorphism** to keep the independency between logic and its implementation

```
public Money calculatePay(Employee e)
    throws InvalidEmployeeType {
    switch (e.type) {
        case COMMISSIONED:
            return calculateCommissionedPay(e);
        case HOURLY:
            return calculateHourlyPay(e);
        case SALARIED:
            return calculateSalariedPay(e);
        default:
            throw new InvalidEmployeeType(e.type);
    }
}
```



```
public abstract class Employee {
    public abstract boolean isPayday();
    public abstract Money calculatePay();
    public abstract void deliverPay(Money pay);
}

public interface EmployeeFactory {
    public Employee makeEmployee(EmployeeRecord r)
        throws InvalidEmployeeType;
}

public class EmployeeFactoryImpl implements EmployeeFactory {
    public Employee makeEmployee(EmployeeRecord r)
        throws InvalidEmployeeType {
        switch (r.type) {
            case COMMISSIONED:
                return new CommissionedEmployee(r);
            case HOURLY:
                return new HourlyEmployee(r);
            case SALARIED:
                return new SalariedEmployee(r);
            default:
                throw new InvalidEmployeeType(r.type);
        }
    }
}
```

Functions

Function Arguments

- ✓ The ideal number of arguments for a function is zero
- ✓ Next comes one, followed closely by two
- ✓ Three arguments should be avoided where possible
- ✓ More than three requires very special justification.

Likely that those arguments should be wrapped into a class of their own

Keep the number as **less as possible**

Functions

Have No Side Effects



Not does other *hidden* things

```
public class UserValidator {
    private Cryptographer cryptographer;

    public boolean checkPassword(String userName, String password) {
        User user = UserGateway.findByName(userName);
        if (user != User.NULL) {
            String codedPhrase = user.getPhraseEncodedByPassword();
            String phrase = cryptographer.decrypt(codedPhrase, password);
            if ("Valid Password".equals(phrase)) {
                Session.initialize();
                return true;
            }
        }
        return false;
    }
}
```

Functions

Don't repeat yourself (DRY)



Don't copy and paste the same code over and over again. Consider the

abstraction!

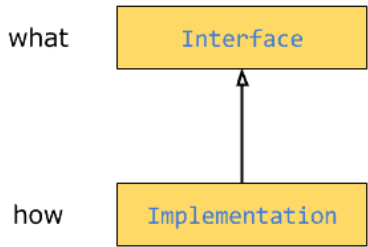


Same algorithm but different codes is still a **duplication**.

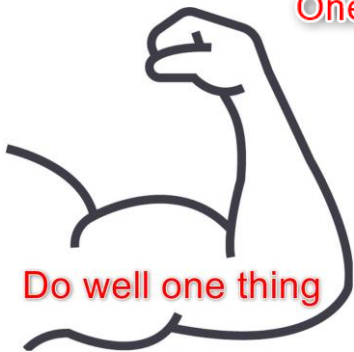
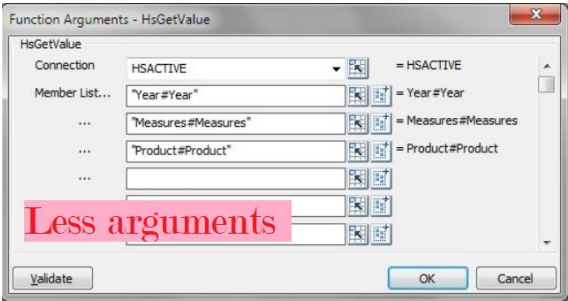
```
1
2 public class CustomerNameChanger {
3     public void ChangeName(CustomerDbContext context, int customerID, string name) {
4         var customer = context.Customer.SingleOrDefault(x => x.customerID == customerID);
5         if(customer == null)
6             throws new Exception(string.Format("Customer {0} was not found.", customerID));
7
8         customer.Name = name;
9     }
10 }
11
12 public class CustomerAddressChanger {
13     public void ChangeAddress(CustomerDbContext context, int customerID, string address,
14                             string postalCode, string city) {
15         var customer = context.Customer.SingleOrDefault(x => x.customerID == customerID);
16         if(customer == null)
17             throws new Exception(string.Format("Customer {0} was not found.", customerID));
18
19         customer.Address = address;
20         customer.PostalCode = postalCode;
21         customer.City = city;
22     }
23 }
```

Functions

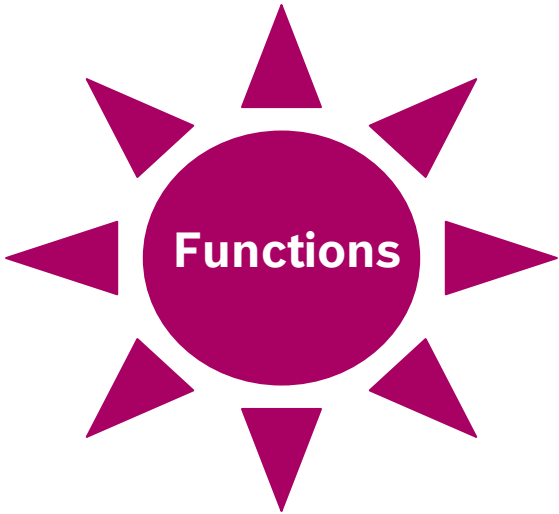
Take away



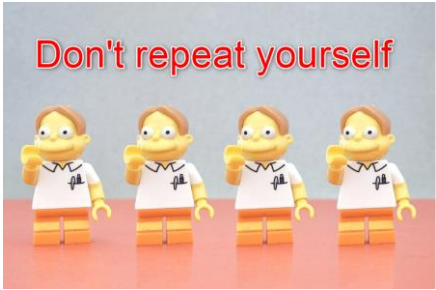
One level Abstraction



Do well one thing

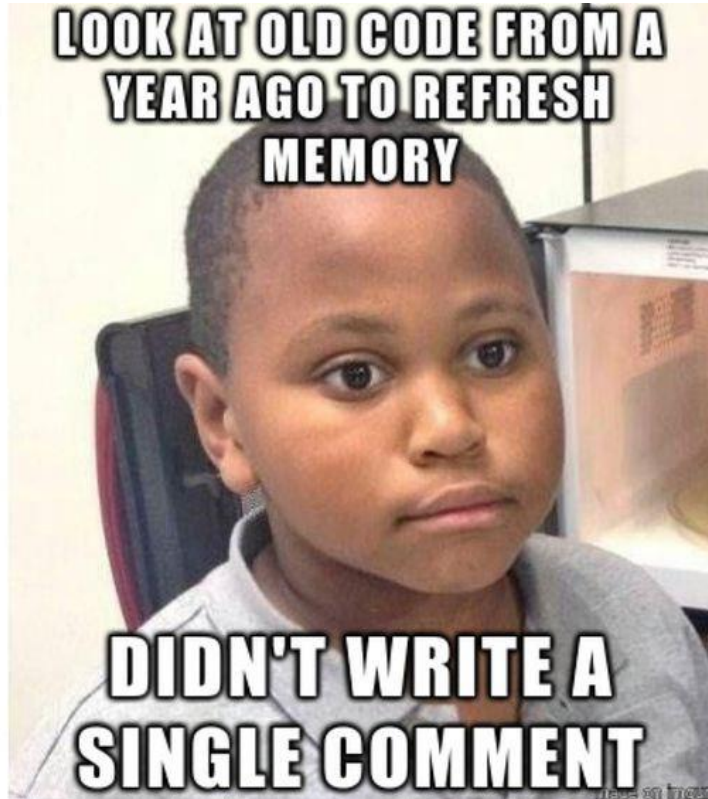


Small and Smaller



COMMENTS

Comments



CODE COMMENTS BE LIKE



Comments

Pros...

```
1
2 private final String HTTP_DATE_REGEX =
3 "[SMTWF][a-z]{2}\\,\\s[0-9]{2}\\s[JFMASO][a-z]{2}\\s"+
4 "[0-9]{4}\\s[0-9]{2}\\:[0-9]{2}\\:[0-9]{2}\\sGMT";
5
6
```



```
1
2 private final String HTTP_DATE_REGEX =
3 "[SMTWF][a-z]{2}\\,\\s[0-9]{2}\\s[JFMASO][a-z]{2}\\s"+
4 "[0-9]{4}\\s[0-9]{2}\\:[0-9]{2}\\:[0-9]{2}\\sGMT";
5 // Example: "Tue, 02 Apr 2003 22:18:49 GMT"
6
```

Nothing can be
quite so **helpful** as a
well-placed
comment

Comments ... and Cons

```
1
2 //Returns x + y or, if x or y is less than zero, throws an exception
3 public int Add(int x, int y)
4 {
5     return x + y;
6 }
7
```

```
1
2 // Utility method that returns when this.closed is true. Throws an exception
3 // if the timeout is reached.
4 public synchronized void waitForClose(final long timeoutMillis)
5 throws Exception
6 {
7     if(!closed)
8     {
9         wait(timeoutMillis);
10        if(!closed)
11            throw new Exception("MockResponseSender could not be closed");
12    }
13 }
14
```

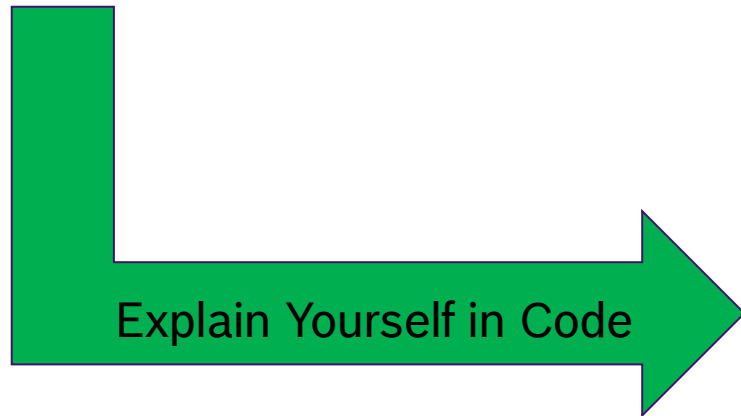


Nothing can be
quite so damaging as a
NOT up-to-date and
Misinformation
comment

Comments

Explain yourself in Code

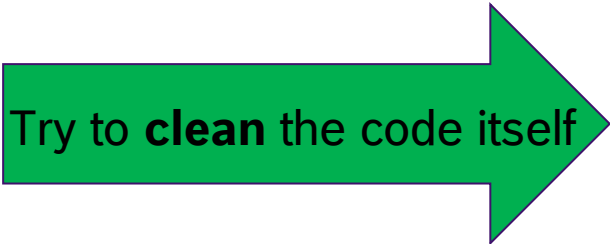
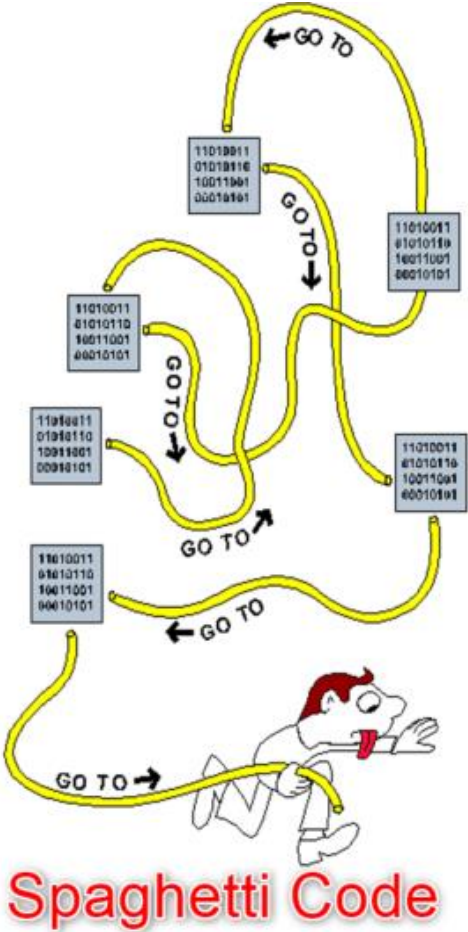
```
1
2
3  if (employee.isFlags() && HOURLY_FLAG && employee.getAge() > 65)
4  {
5      |    ...
6  }
7
```



```
1
2  if (employee.isEligibleForFullBenefits())
3  {
4      |    ...
5  }
6
```


Comments

Comments for Bad code



Good comments

- ✓ Legal comments
- ✓ **Informative** comments
- ✓ Explanation of **Intent**
- ✓ Clarification
- ✓ Warning of **Consequences**
- ✓ **TODO** comments
- ✓ Amplification

```
// format matched kk:mm:ss EEE, MMM dd, yyyy
Pattern timeMatcher = Pattern.compile(
    "\\d*:\\d*:\\d* \\w*, \\w* \\d*, \\d*");
```

```
//This is our best attempt to get a race condition
//by creating large number of threads.
for (int i = 0; i < 25000; i++) {
    ...
    thread.start();
}
```

```
public static SimpleDateFormat makeStandardHttpDateFormat()
{
    //SimpleDateFormat is not thread safe,
    //so we need to create each instance independently.
    SimpleDateFormat df = new SimpleDateFormat("EEE, dd MMM yyyy HH:mm:ss z");
    df.setTimeZone(TimeZone.getTimeZone("GMT"));
    return df;
}
```

Bad comments

✗ **Misleading** comments

✗ **Journal** comments

✗ **Redundant** comments

✗ **Noise** comments

✗ **Nonlocal** Information

✗ **Commented-out** code

```
// Changes (from 11-Oct-2001)
// -----
// 11-Oct-2001 : Re-organised the class and moved it to new package
// 03-Oct-2002 : Fixed errors reported by Checkstyle;
// 13-Mar-2003 : Implemented Serializable;
// 29-May-2003 : Fixed bug in addMonths method;
```

```
function DISCOUNT (quantity, price) {
    //Calculate a discount for items greater than 100 units
    if quantity >= 100                // Check if it's more than 100 units
    |   DISCOUNT = quantity * price * 0.1;    // Calculate a 10% discount
    else
    |   DISCOUNT = 0;                        // Otherwise no discount
    end if;

    DISCOUNT = Application.Round(DISCOUNT,2); // Round to 2 decimal places
}
```

```
// Port on which fitnessse would run. Defaults to <b>8082</b>.
//   @param fitnesssePort
public void setFitnesssePort(int fitnesssePort) {
    |   this.fitnesssePort = fitnesssePort;
}
```

Comments

Take away



Rather than spend your time writing the **comments** that explain the mess you've made, spend it **cleaning** that mess.

Guru programmers tell you **why** other implementations were **not** chosen.

Great programmers tell you **why** a particular implementation was chosen.

Good programmers **comment** their code.

FORMATTING

What do you see
in this picture?



The **broken** window theory



Image source: <http://artnerdy.blogspot.com/2014/12/broken-window-theory.html>

Power of formatting



Hmm... No one cares about how to write an easy-to-read code here. The other modules of this project will be like this only!!!

```
function addlist(i){
  document.getElementById("addlist").addEventListener("click",
    function(){
      var color = "green";
      if (document.getElementById("checked").checked) {
        color = "red";
      }
      var x1 = document.getElementById("x1").value;
      var y1 = document.getElementById("y1").value;
      var x2 = document.getElementById("x2").value;
      var y2 = document.getElementById("y2").value;
      console.log(`${x1} ${y1} ${x2} ${y2}`);
      var c = document.getElementById("canvas");
      var ctx = c.getContext("2d");
      ctx.beginPath();
      ctx.moveTo(x1,y1);
      ctx.lineTo(x2, y2);
      ctx.stroke();}
  );
}
```


Power of formatting

- ❖ Code formatting is about **communication**
... and communication is **first need** of
a professional developer



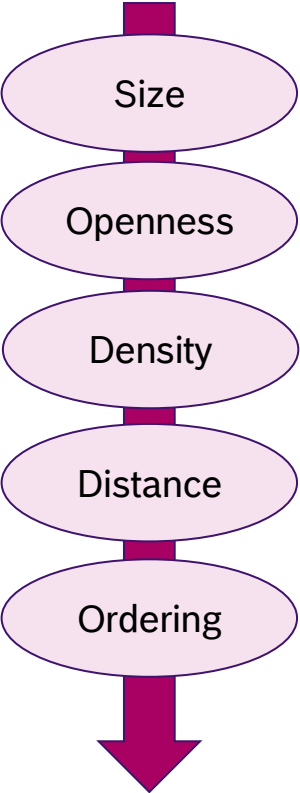
- ❖ The code itself might be changed in the future
... but the **coding style** and **readability**
will **live** together with the project.

```
function register()
{
    if (isEmpty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if (isset($_POST['user_password_new'])) {
                if (isset($_POST['user_password_request'])) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 45 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/[a-zA-Z0-9,44@!]/', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if (isset($_POST['user_email'])) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 44 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Password must match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```

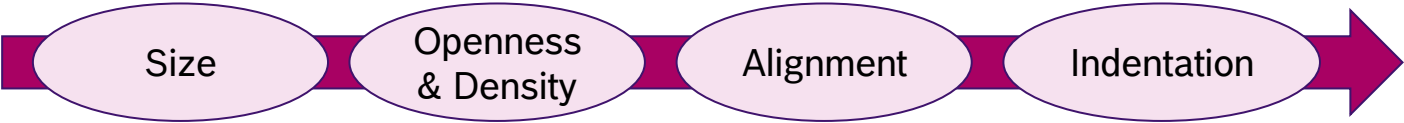
A small illustration of a karate master in a white gi, standing in a ready stance. This image is placed next to the code block to represent the concept of 'coding style'.

Formatting

Vertical
Formatting



```
10 -
11 def generate_sequences(n, length=3, prop_no_change=0.2):
12     """Short summary.
13     """
14     Parameters
15     -----
16     n : type
17     Description of parameter 'n'.
18     length : type
19     Description of parameter 'length'.
20     prop_no_change : type
21     Description of parameter 'prop_no_change'.
22     Returns
23     -----
24     type
25     Description of returned object.
26     """
27     """
28     """
29     x_range = [0, lc_all.width - 1]
30     y_range = [0, lc_all.height - 1]
31     all_samples = []
32     max_no_change = round(n * prop_no_change)
33     max_changed = n - max_no_change
34     n_changed = 0
35     n_no_change = 0
36     while len(all_samples) < n:
37         while True:
38             x = random.randrange(*x_range)
39             y = random.randrange(*y_range)
40             win = Window(x, y, randrange(self.start, stop=None, step=1, _int=int))
41             which = range(lc_a
42             sample = np.array( Choose a random item from range(start, stop, step))
```



Horizontal Formatting

Vertical Formatting

✓ Vertical **Size**: How big should a source file be?

→ Typically **200** lines long, upper limit of 500

✓ Vertical **Openness** between concepts

Add blank line

```
1 package fitness.wikitext.widgets;
2
3
4 import java.util.regex.*;
5
6 public class BoldWidget extends ParentWidget {
7     public static final String REGEXP = "''(.+?)''";
8     private static final Pattern pattern = Pattern.compile("''(.+?)''",
9         Pattern.MULTILINE + Pattern.DOTALL
10    );
11
12    public BoldWidget(ParentWidget parent, String text) throws Exception {
13        super(parent);
14        Matcher match = pattern.matcher(text);
15        match.find();
16        addChildWidgets(match.group(1));
17    }
18
19    public String render() throws Exception {
20        StringBuffer html = new StringBuffer("<b>");
21        html.append(childHtml()).append("</b>");
22        return html.toString();
23    }
24 }
```

Vertical Formatting

✓ Vertical **Size**: How big should a source file be?

→ Typically **200** lines long, upper limit of 500

✓ Vertical **Openness** between concepts

✓ Vertical **Density** within concept

Keep dense

```
1 package fitnessse.wikitext.widgets;
2
3
4 import java.util.regex.*;
5
6 public class BoldWidget extends ParentWidget {
7     public static final String REGEXP = "''.+?'";
8     private static final Pattern pattern = Pattern.compile("''.+?'",
9         Pattern.MULTILINE + Pattern.DOTALL
10    );
11
12    public BoldWidget(ParentWidget parent, String text) throws Exception {
13        super(parent);
14        Matcher match = pattern.matcher(text);
15        match.find();
16        addChildWidgets(match.group(1));
17    }
18
19    public String render() throws Exception {
20        StringBuffer html = new StringBuffer("<b>");
21        html.append(childHtml()).append("</b>");
22        return html.toString();
23    }
24 }
```

Vertical Formatting

- ✓ Vertical **Size**: How big should a source file be?
 - Typically **200** lines long, upper limit of 500
- ✓ Vertical **Openness** between concepts
- ✓ Vertical **Density** within concept
- ✓ Vertical **Distance**: Related concepts should be closed to each other
- ✓ Vertical **Ordering**: The called function should be kept before/after the calling function

Horizontal Formatting



Horizontal **Size**: How wide should a line be?

→ Typically **80 ~ 120** chars/line, **NEVER** have to scroll to the right



Horizontal **Openness**

Add space

```
1
2 public class Quadratic {
3     public static double root1(double a, double b, double c) {
4         double determinant = determinant(a, b, c);
5         return (-b + Math.sqrt(determinant)) / (2*a);
6     }
7     public static double root2(int a, int b, int c) {
8         double determinant = determinant(a, b, c);
9         return (-b - Math.sqrt(determinant)) / (2*a);
10    }
11    private static double determinant(double a, double b, double c) {
12        return b*b - 4*a*c;
13    }
14 }
```

Horizontal Formatting



Horizontal **Size**: How wide should a line be?

→ Typically 80 ~ 120 chars/line, **NEVER** have to scroll to the right



Horizontal **Openness**



Horizontal **Density**

Keep dense

```
1
2 public class Quadratic {
3     public static double root1(double a, double b, double c) {
4         double determinant = determinant(a, b, c);
5         return (-b + Math.sqrt(determinant)) / (2*a);
6     }
7     public static double root2(int a, int b, int c) {
8         double determinant = determinant(a, b, c);
9         return (-b - Math.sqrt(determinant)) / (2*a);
10    }
11    private static double determinant(double a, double b, double c) {
12        return b*b - 4*a*c;
13    }
14 }
```

Horizontal Formatting



Horizontal **Alignment**

```
1
2 public class FitNesseExpediter implements ResponseSender
3 {
4     private Socket socket;
5     private InputStream input;
6     private OutputStream output;
7     private Request request;
8     private Response response;
9     private FitNesseContext context;
10    protected long requestParsingTimeLimit;
11    private long requestProgress;
12    private long requestParsingDeadline;
13    private boolean hasError;
14
15    public FitNesseExpediter(Socket s,
16                             | | | | | FitNesseContext context) throws Exception
17    {
18        this.context = context;
19        socket = s;
20        input = s.getInputStream();
21        output = s.getOutputStream();
22        requestParsingTimeLimit = 10000;
23    }
```

```
1
2 public class FitNesseExpediter implements ResponseSender
3 {
4     private Socket socket;
5     private InputStream input;
6     private OutputStream output;
7     private Request request;
8     private Response response;
9     private FitNesseContext context;
10    protected long requestParsingTimeLimit;
11    private long requestProgress;
12    private long requestParsingDeadline;
13    private boolean hasError;
14
15    public FitNesseExpediter(Socket s, FitNesseContext context) throws Exception
16    {
17        this.context = context;
18        socket = s;
19        input = s.getInputStream();
20        output = s.getOutputStream();
21        requestParsingTimeLimit = 10000;
22    }
```


Horizontal Formatting



Indentation

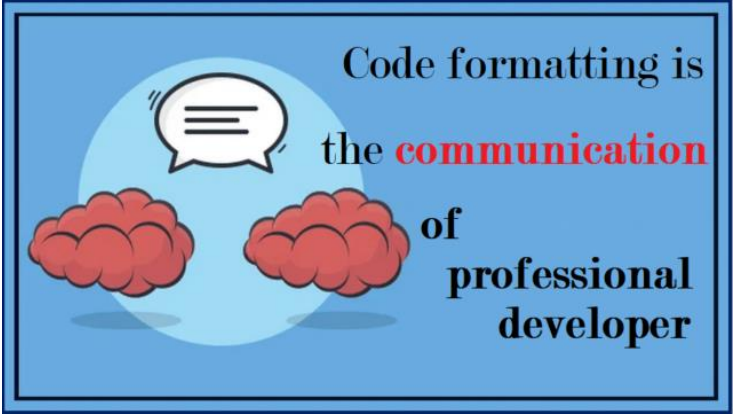
```
1 public class FitNesseServer implements SocketServer { private FitNesseContext
2 context; public FitNesseServer(FitNesseContext context) { this.context =
3 context; } public void serve(Socket s) { serve(s, 10000); } public void
4 serve(Socket s, long requestTimeout) { try { FitNesseExpediter sender = new
5 FitNesseExpediter(s, context);
6 sender.setRequestParsingTimeLimit(requestTimeout); sender.start(); }
7 catch(Exception e) { e.printStackTrace(); } } }
```



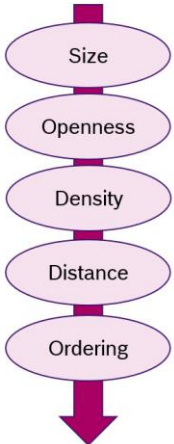
```
1 public class FitNesseServer implements SocketServer {
2     private FitNesseContext context;
3
4     public FitNesseServer(FitNesseContext context) {
5         this.context = context;
6     }
7
8     public void serve(Socket s) {
9         serve(s, 10000);
10    }
11
12    public void serve(Socket s, long requestTimeout) {
13        try {
14            FitNesseExpediter sender = new FitNesseExpediter(s, context);
15            sender.setRequestParsingTimeLimit(requestTimeout);
16            sender.start();
17        }
18        catch (Exception e) {
19            e.printStackTrace();
20        }
21    }
22 }
23
24 }
```

Formatting

Take away



Vertical Formatting



```
def generate_sequences(n, length=3, prop_no_change=0.2):  
    """short summary.  
    Parameters  
    -----  
    n : type  
        Description of parameter 'n'.  
    length : type  
        Description of parameter 'length'.  
    prop_no_change : type  
        Description of parameter 'prop_no_change'.  
    Returns  
    -----  
    type  
        Description of returned object.  
    """  
    x_range = [0, lc_all.width - 1]  
    y_range = [0, lc_all.height - 1]  
    all_samples = []  
    max_no_change = round(n * prop_no_change)  
    max_changed = n - max_no_change  
    n_changed = 0  
    n_no_change = 0  
    while len(all_samples) < n:  
        while True:  
            x = random.randrange(*x_range)  
            y = random.randrange(*y_range)  
            win = Window(x, y, randrange(1, start=None, stop=1, _int=int))  
            which = range(lc_a  
            result = win.get_content() (Choose a random item from content and doof stuff)
```



Horizontal Formatting



Writing Code Game

- Divide into four groups: Three groups as the competitors and one group as the referee
- You are provided a sample page of coding (in C programming language) to identify as much as possible the code smells in the sample and correct them (15 minutes)
- The referee group will show and explain all the code smells to the competitors (15 minutes).
- Each identified code smell will get 5 points, which group have largest total points will be the **WINNER** of the game.